Flight-Project

By: Alex Bringuel, Camilo Monsalve, and Aisha Patel

# Requirements and Specifications

When the project was presented, there were three main requirements that stood out. The first was a working GUI. The second was a database connection to the Java code. The third was proper Java syntax and classes. Knowing this and everything else in between, the project was going to require a lot of detail and care.

GUI requirements (JavaFX):

1. Login page
2. Registration page
3. Reset Password page
4. Main Menu page (Landing page)
5. Add flights page
6. Admin page
7. Add/Edit/Delete Flights page(s)
8. Add/Edit/Delete Admins page(s)

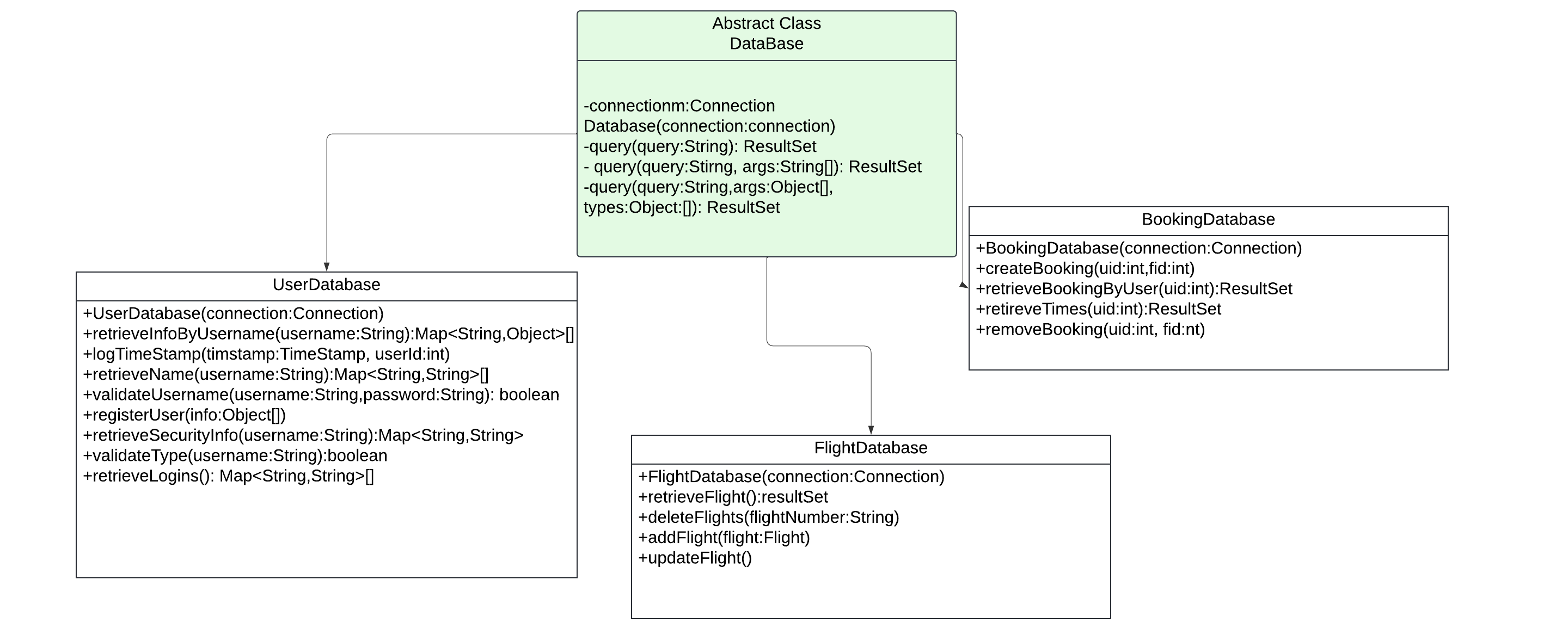
Database requirements (Azure and MySQL):

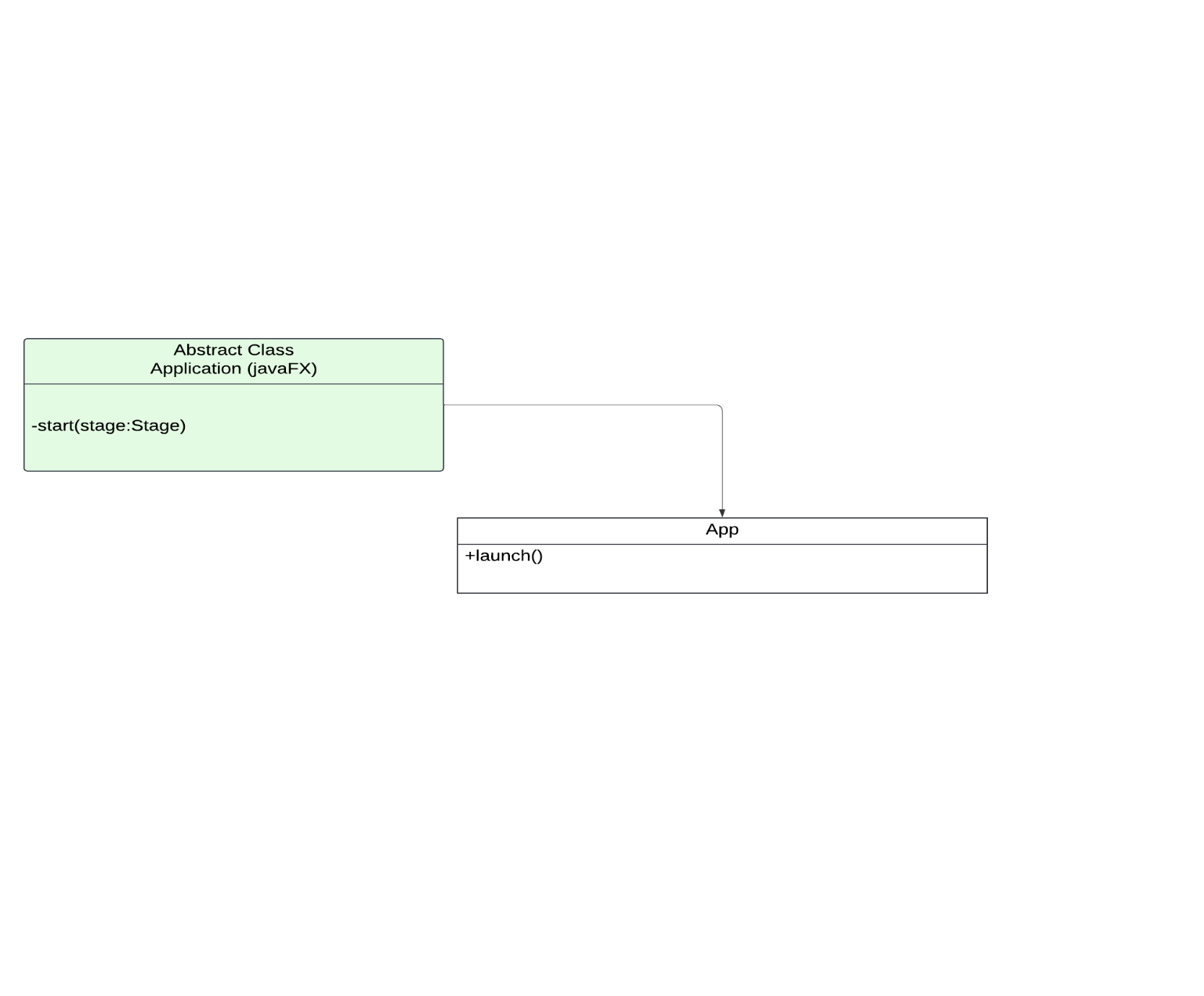
1. Users table (Includes both regular users and administrators)
   1. PK will hold special identification for all user ids (int serial)
2. Flights table (Include all flights)
   1. PK will hold special identification for all flights (int serial)
3. Bookings table
   1. PK will hold special identification for all booking ids (int serial)
   2. FK User ID
   3. FK Flight number
4. Azure server to store data and keep live while application is running

Coding requirements (Java)

1. GitHub repository to share and collaborate with for code
2. Maven for Java FX and SQL
3. Base classes for repetitive methods and abstraction
4. Interfaces and/ Abstraction
5. Encapsulation
6. Polymorphism
7. Composition

# Class UML

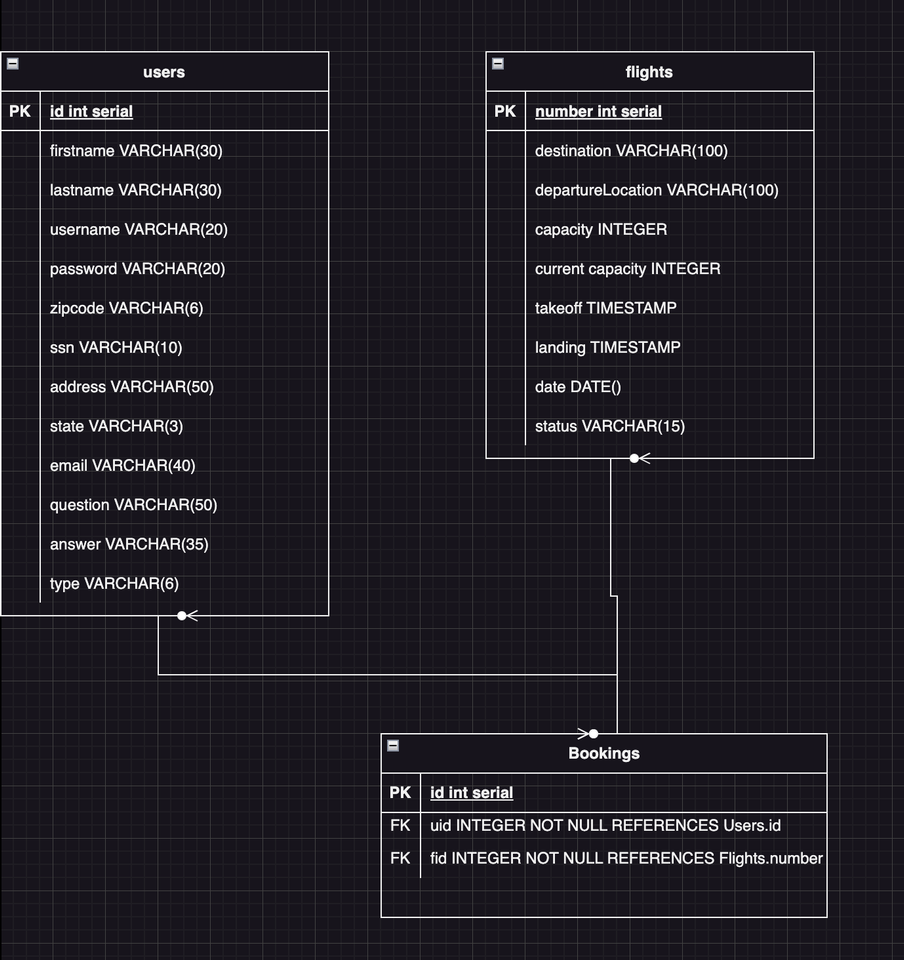




In our class UML we highlighted three main abstract classes: Screen, DataBase, and Application.

1. The **Screen** class has an abstract method where the user must use the method to return a GridPane. This was useful in creating the UI, because not only did it clean up the code and allowed for consistency, but it also kept the main GridPane code in one block. This class also had a non-abstract method called createScreen. This requires the class to return the screen in its designated style.
2. The **DataBase** class gets a little tricky and intricate. There are multiple things to note. Firstly, there is a private final connection variable alongside a construction that defines the connection. After this, there are a long list of protected methods that run through the query and return a result set, as seen on the UML. These methods are not abstract and are called when needed based on the subclass.
3. The last abstract class we investigated was the **Application** class. This one was given to us by JavaFX. We used this class to initiate the start method to launch our argument.

# Data Model



Our data model highlights three tables. We decided to keep all the admin and user information in the users table to keep things simple. Throughout the app, the user table will select, insert, update, and delete based on the choices made by the users on the UI. The flights table will hold all the flight information and will be changed through an administrative account. This account will have the ability to select, update, insert, and delete the flight information. Lastly, the bookings table will query the flights and customer information together.

# Functionality

The first package we have is the animation package where we code some animations for the UI. This class will be called where needed. The next package is vital to the project. It is the bases folder/. This package includes the component file which will be abstracted to the button classes and card classes, and the database class that is the superclass for the database files. Additionally, the models class, the navigate class that is abstracted to the PushEditFlight class. We also have the panes and screen class which are abstracted to all the screens and panes. Our components package holds the buttons that will hold our main buttons, such as main menu and logout. The cards package holds the cards for flightcard and userinfocard. These classes hold the design of the card information that will show up on the UI. We have an input field class that creates an object of an input field with certain sizes and parameters. Lastly we have our app class which launches all these files and creates the maven project.